Operating Systems Laboratory

Lab 7

Om Patil (200010036)

Q1.

1. Seed = 1
   Base = 0x0000363c (decimal 13884), Limit = 290

Virtual Address	In Bounds?	Translation
0x0000030e (decimal: 782)	No	
0x00000105 (decimal: 261)	Yes	0x00003741 (decimal: 14145)
0x000001fb (decimal: 507)	No	
0x000001cc (decimal: 460)	No	
0x0000029b (decimal: 667)	No	

Seed = 2
Base = 0x00003ca9 (decimal 15529), Limit = 500

Virtual Address	In Bounds?	Translation
0x00000039 (decimal: 57)	Yes	0x00003ce2 (decimal: 15586)
0x00000056 (decimal: 86)	Yes	0x00003cff (decimal: 15615)
0x00000357 (decimal: 855)	No	
0x000002f1 (decimal: 753)	No	
0x000002ad (decimal: 685)	No	

Seed = 3
Base = 0x000022d4 (decimal 8916), Limit = 316

Virtual Address	In Bounds?	Translation

0x0000017a (decimal: 378)	No	
0x0000026a (decimal: 618)	No	
0x00000280 (decimal: 640)	No	
0x00000043 (decimal: 67)	Yes	0x00002317 (decimal: 8983)
0x0000000d (decimal: 13)	Yes	0x000022e1 (decimal: 8929)

2. The limit must be set to 930 to ensure all the generated virtual addresses are within bounds.
3. The maximum value that the base can be set to, such that the address space still fits into physical memory in its entirety, is 16284.
4. Considering address space of size 4k and physical memory of size 128k, the answer to questions 1-3 are as follows,

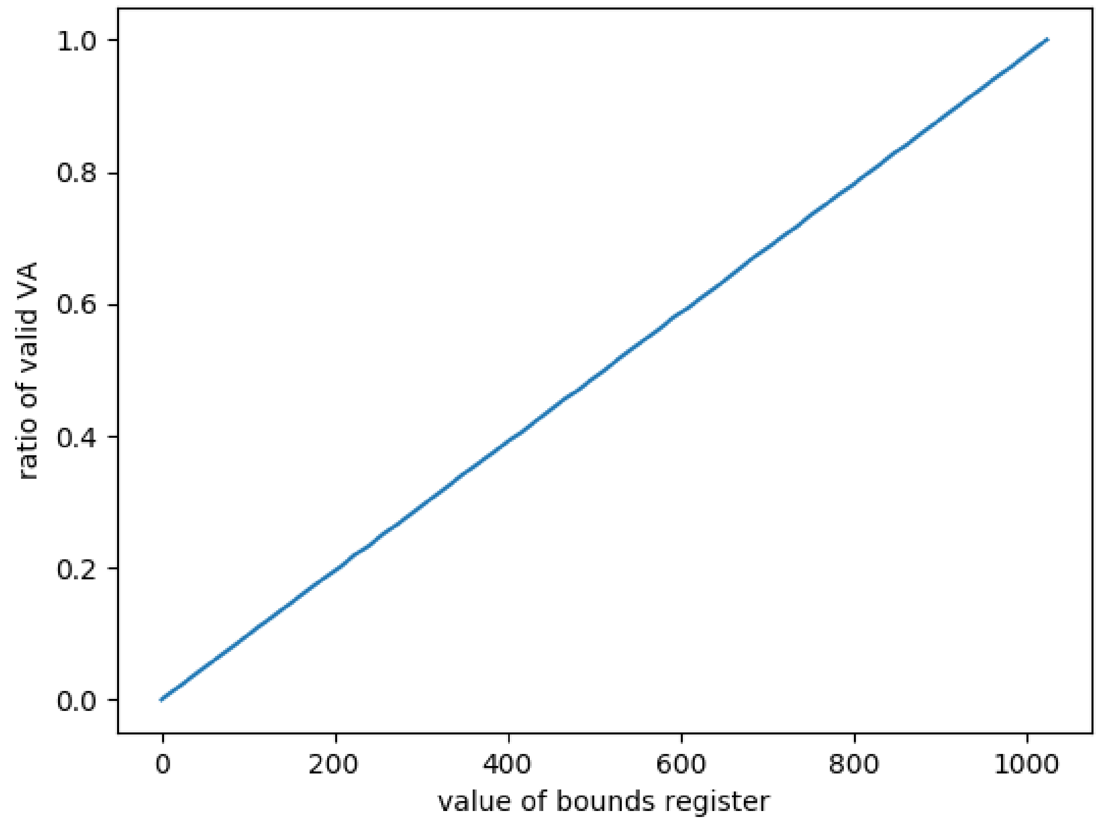
   Seed = 1
   Base = 0x0000363c (decimal 13884), Limit = 290

Virtual Address	In Bounds?	Translation
0x00000c38 (decimal: 3128)	No	
0x00000414 (decimal: 1044)	Yes	0x0001b5f6 (decimal: 112118)
0x000007ed (decimal: 2029)	No	
0x00000731 (decimal: 1841)	No	
0x00000a6c (decimal: 2668)	No	

   Seed = 2
   Base = 0x0000363c (decimal 13884), Limit = 290

Virtual Address	In Bounds?	Translation
0x00000c38 (decimal: 3128)	No	
0x00000414 (decimal: 1044)	Yes	0x0001b5f6 (decimal: 112118)
0x000007ed (decimal: 2029)	No	
0x00000731 (decimal: 1841)	No	
0x00000a6c (decimal: 2668)	No	

5. ratio of valid VA = value of bounds register / size of address space



Q2.

1. `segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0`

Virtual Address	Valid?	Translation
0x0000006c (decimal: 108)	Yes	0x000001ec (decimal: 492)
0x00000061 (decimal: 97)	No	
0x00000035 (decimal: 53)	No	
0x00000021 (decimal: 33)	No	
0x00000041 (decimal: 65)	No	

```
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
```

Virtual Address	Valid?	Translation
0x00000011 (decimal: 17)	Yes	0x00000011 (decimal: 17)
0x0000006c (decimal: 108)	Yes	0x000001ec (decimal: 492)
0x00000061 (decimal: 97)	No	
0x00000020 (decimal: 32)	No	
0x0000003f (decimal: 63)	No	

```
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
```

Virtual Address	Valid?	Translation
0x0000007a (decimal: 122)	Yes	0x000001fa (decimal: 506)
0x00000079 (decimal: 121)	Yes	0x000001f9 (decimal: 505)
0x00000007 (decimal: 7)	Yes	0x00000007 (decimal: 7)
0x0000000a (decimal: 10)	Yes	0x0000000a (decimal: 10)
0x0000006a (decimal: 106)	No	

2. The highest legal virtual address in segment 0 is 19. The lowest legal address in segment 1 is 108. We can verify this by passing the -A 19,20,108,107 option.
3. We would set the following base and bounds,
   ```
   segmentation.py -a 16 -p 128 -A
   0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 0 --l0 2 --b1 15 --l1
   2
   ```
4. To get roughly 90% of the virtual addresses generated to be valid, we should set the sum of the two limits to be 0.9 times the address space. The bounds should be such that their difference is larger than the sum of the limits, i.e. the two segments do not overlap.
5. Yes, the simulator can be run such that no virtual addresses are valid. This can be done by setting both the limit value to 0.

Q3.

1. `paging-linear-size.py -v 32 -p 256 -e 4`

   Size = 67108864 Bytes

   No. of Offset Bits = log(256) = 8
   No. of Virtual Page Number Bits = 32 - 8 = 24
   No. of Entries in Page Table = 2^24 = 16777216

   Size of Page Table = 16777216*4 = 67108864

2. `paging-linear-size.py -v 16 -p 128 -e 8`

   Size = 4096 Bytes

   No. of Offset Bits = log(128) = 7
   No. of Virtual Page Number Bits = 16 - 7 = 9
   No. of Entries in Page Table = 2^9 = 512

   Size of Page Table = 512*8 = 4096

3. `paging-linear-size.py -v 12 -p 4 -e 32`

   Size = 32768 Bytes

   No. of Offset Bits = log(4) = 2
   No. of Virtual Page Number Bits = 12 - 2 = 10
   No. of Entries in Page Table = 2^10 = 1024

   Size of Page Table = 1024*32 = 32768

Q4.

1. `paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0`
   Size = 1024

   `paging-linear-translate.py -P 1k -a 2m -p 512m -v -n 0`
   Size = 2048

   `paging-linear-translate.py -P 1k -a 4m -p 512m -v -n 0`
   Size = 4096

As seen in the above test cases linear page table size scales linearly with respect to address space size.

```
paging-linear-translate.py -P 1k -a 1m -p 512m -v -n 0
```
Size = 1024

```
paging-linear-translate.py -P 2k -a 1m -p 512m -v -n 0
```
Size = 512

```
paging-linear-translate.py -P 4k -a 1m -p 512m -v -n 0
```
Size = 256

As seen in the above test cases linear page table size scales inversely with respect to page size.

Big page sizes lead to a lot of unused memory space as only a fraction of the large page is actually used, but the entire page is allocated.

2. As we increase the percentage of pages that are allocated in each address space, more and more entries in the page table get filled.

3. The first parameter option is unrealistic as the size is too small for practical use cases.

4.
    a. The address space cannot be larger than the physical memory, as there would not be enough space to fit all the available addresses.
    b. The address space size and physical memory size must be non-zero as else it would not be possible to simulate anything.
    c. The simulator only supports physical memory sizes below 1 GB, as anything larger would be quite computationally expensive to simulate.
    d. The address space and physical memory size must be a power of 2, else an integer number of bytes will not fit inside them.
    e. The address space and physical memory size must be a multiple of page size, else an integer number of pages will not fit inside them.